# Efficient AES Implementation for Better Resource Usage and Performance of IoTs

Umer Farooq
Department of Electrical Engineering
Dhofar University
Salalah, Oman
Email: ufarooq@du.edu.om

Maria Mushtaq
LIRMM-CNRS
Univ Montpellier
Montpellier, France
Email: maria.mushtaq@lirmm.fr

Muhammad Khurram Bhatti
Department of Computer Engineering
Information Technology University
Lahore, Pakistan
Email: khurram.bhatti@itu.edu.pk

***Abstract*—The research on Internet of Things (IoT) devices has advanced tremendously over the past few years. IoT-based systems have their applications in almost every sphere of human life. Modern IoT devices are of quite heterogeneous nature and they are going to be involved in every thing from turning home lights ON/OFF to handling life critical data of a patient in smart health system. Because of the amount and nature of the data handled by IoT devices, they are a lucrative target for various kinds of security attacks. Among the many countermeasures against the security threats, Advanced Encryption Standard (AES) is a popular cryptographic scheme as it offers robust and platform independent implementation. In this work, keeping in view of the heterogeneous nature of the target IoT devices, we explore five different implementations of AES algorithm. These implementations use different algorithmic and architecture optimizations. The results obtained through these implementations reveal that some of them are very suitable for resource constrained edge IoT devices while others are useful for performance hungry middle layer gateways of an IoT-based system. Experimental results reveal that in an IoT-based system, a uniform cryptographic implementation should not be considered and that the implementations should be altered as per the nature of the target device.**

***Keywords–Cryptography; Embedded Security; AES.***

## I. INTRODUCTION

Over the past few years, the research in embedded systems, especially their miniaturized form i.e. the Internet of Things (IoTs) devices has made huge progress. The IoT devices are going to be part of human life for a long time to come and they are speculated to change the way humans perceive about their life [1]. The IoT devices powered by high performance embedded systems have their applications in almost every sphere of human life like smart vehicles, homes, health systems, environmental monitoring, supply chains, etc. [2]. The aforementioned applications of IoT devices indicate that they have to handle enormous amounts of critical data. The handling of the data means processing, transmission, and storage of data. The critical nature of the data handled by IoT devices makes them a lucrative target for potential security attacks. The potential security threats can result in the compromise of integrity and the availability of data. A compromised IoT-based system can put even lives in danger [3] [4]. Hence, a secured IoT-based system, where the integrity and authenticity of the data is ensured through proper security measures becomes of paramount importance [5].

The importance of security for an IoT-based system is clear from the discussion presented above. However, there is no one standard way to make an IoT-based system secure [6]. This is because of the fact the IoT-based systems are normally multi-layered systems and different layers require different measures to make them secure. For example, the top layers like application and network layer are made secure easily through well established firewalls and security protocols. But the security of edge side layer is a hugely challenging task because of the varying nature of the edge side IoT devices and different types of security threats [7]. The edge side nodes of an IoT-based system are normally quite heterogeneous in nature. They have different hardware resources with varying performance requirements. These nodes are normally subjected to a range of security attacks like hardware trojans, side channel attacks, denial of service attacks [8]. All the aforementioned attacks compromise the authenticity, integrity, or the availability of the data in an IoT-based system. A number of countermeasures to these attacks like side channel analysis, isolation, blocking, and implementation of cryptographic algorithms have been proposed in the past [9] [10]. Among these countermeasures, the cryptographic schemes are of particular importance as they offer robust and hardware independent solutions. There are many cryptographic schemes that have been used in the past to secure embedded systems. Some of the most commonly used techniques include Data Encryption Standard (DES), 3-DES, and Advanced Encryption Standard (AES) techniques. AES is a cryptographic technique that uses symmetric cipher and offers highest possible security level. Standard implementation of AES on the hardware is quite challenging in terms of resource and performance requirements and it is not normally suited for resource constrained and performance critical IoT devices.

A lot of work has been done in the recent past to improve the efficiency of AES algorithm in embedded system. The improvement in efficiency means mainly reduced resource requirement with improved performance. Most of the work in the state-of-the-art considers the implementation of AES on FPGA. For example, the authors in [11] implement the AEs algorithm in a completely sequential manner. The sequential implementation results in a design that requires fewer resources as compared to existing solutions and this kind of implementation is well suited for resource constrained embedded systems. Similarly, the authors in [12] present the power efficient implementation of AES algorithm that is well suited for power constrained devices. Authors in [13] present another efficient implementation of AES that uses concepts of loop unrolling and parallelism to achieve high performance. This kind of implementation is well suited for applications requiring high speed and where the resources are not a

 ISBN: 978-1-61208-818-1 

constraint. Moreover, authors in [14]–[16] explore further high speed implementations of AES algorithm that provide very low delay numbers but require high number of resources on the target architecture. Although these implementations give good results in terms pf performance, they are not well suited for resource constrained IoT devices as they require huge amount of resources. To address this problem, authors in [17] present a version of AES implementation that is well suited for resource constrained IoT devices. Moreover, the authors in [9] propose another version of AES implementation that is well suited for IoT devices. It is important to mention here that both aforementioned works target only a single AES implementation and they do not take into account the heterogeneous nature of IoT devices. This kind of implementations might be useful in certain scenarios. However, this kind of static approach cannot be applied across a group of heterogeneous devices.

In this work, we explore five different implementation techniques of AES algorithm. We apply different types of optimizations and based on those optimizations, we obtain different area and performance results for each technique. For example, some of the proposed techniques require very small resources which is very suitable for resource constrained IoT devices. However, their execution speed is also low. On the other hand, there are some other techniques which have very high performance and they can satisfy the requirements of performance constrained IoT devices. But at the same time, they require higher number of resources as well. So, the main contribution of this paper is the provision of a pool of AES implementation techniques that are well suited for the target IoT device. The implementation results of the proposed techniques show that by carefully optimizing the algorithms and by exploiting the resources of target architecture, better area and speed results can be obtained. In the remainder of the paper, Section II gives an overview of AES encryption algorithm. Section III discusses the five implementation techniques and also highlights how these techniques can result in good area and delay trade-offs. Experimental results are discussed in Section IV and the paper is finally concluded in Section V.

## II. Overview of AES Algorithm

AES was selected by National Institute of Standards and Technology (NIST) [18] as a replacement of old DES. The main reason behind its selection was its agility and simple iplementation. At the same time, it provided robust security against all kinds of security threats. AES is an iterative algorithm that is implemented over multiple rounds and it supports key sizes of 128, 192, and 256 bits. Larger the key size, better the security. However, larger key sizes require more resources. In this work, we focus on the AES implementations with 128 bit key size. However, the results obtained with this key size can be extrapolated to larger key sizes as well. In the remaining part of this section, an overview of the AES algorithm is presented.

An overview of the implementation of AES algorithm is shown in Figure 1. It can be seen from this figure that AES implementation in hardware can mainly be divided into two parts: one is called the cipher module and the other is called the key expansion module. Both modules run in parallel where key expansion module generates the key and the cipher module uses that key to encrypt or decrypt the text under consideration. Normally, for 128 bit key size implementation, cipher module performs 10 rounds of operations. In the first nine rounds,

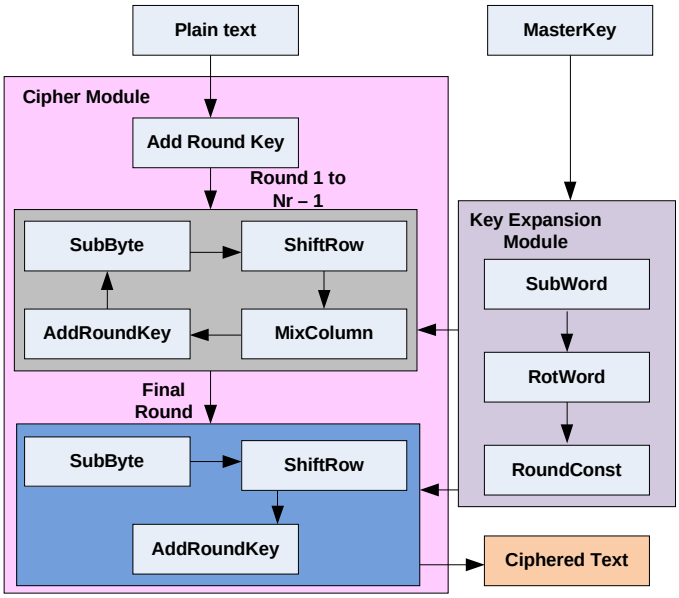


Figure 1. Standard Overview of AES Algorithm

the cipher module performs SubByte, ShiftRow, MixColumn, and AddRoundKey operations. In the final round, MixColumn operation is removed and only SubByte, ShiftRow, and AddRoundKey operations are performed. During each of the ten rounds, key expansion module provides the cipher module with the expanded key through its SubWord, RotWord, and RoundConst operations. It can be seen from Figure 1 that the AES algorithm acts on the input data in an iterative manner to give the encrypted data. Further discussion on the individual operations of two modules of AES algorithm is provided next.

It can be seen from Figure 1 that the cipher module starts with ***SubByte*** operation. This operation takes the input data one byte at a time and replaces it with a byte from the substitution box (also called as S-box). S-box is constructed through two transformations. In the first transformation, multiplicative inverse is taken while in the second part, affine transformation is performed. In this transformation, the input data is multiplied by constant matrix M and the result is then added to an eight bit vector C given below.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, C = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

After the SubByte operation, as the name implies, the ***ShiftRow*** operation performs the circular shift on the rows of the input data. It is important to mention here that the input data is represented as 4x4 matrix where each entry is a byte. The ShiftRow operation performs circular shifting on last three rows while leaving the first row unchanged. This function rotates the second row by one byte, third row by two bytes, and fourth row by three bytes.

TheShiftRow operation operates on the rows of input data whereas the ***MixColumn*** function operates on each of the four columns of the input data. In this function, each column of the

input data is considered as polynomials and given by

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (1)$$

Finally, the ***AddRoundKey*** is the operation that mixes the key with the data using a bit-wise XOR operation and gives the output of the round.

As described earlier, the main purpose of key expansion module is to give the expanded key for each round. In this module, ***SubWord*** function applies S-box to perform substitution and give the output. The ***RotWord*** function performs cyclic permutation and ***RoundConst*** performs bit-wise XOR operation.

## III. Proposed AES Algorithm Implementation

It is clear from the discussion presented in Section II that the implementation of AES algorithm can mainly be divided into two parts: one is the implementation of cipher module and the other is the implementation of key expansion module. The cipher module is mainly an iterative process and its implementation can be paralleled by applying the concept of loop unrolling. In loop unrolling, the N iteration of cipher module are unrolled and they are executed in parallel. The parallelism obtained in cipher module is further aided through splitting in the key expansion module. So, the loop unrolling in cipher module and splitting in key expansion module completely parallelize the implementation of AES algorithm on hardware. This parallel implementation of AES algorithm has the potential to significantly increase the performance of AES algorithm. However, it can also severely increase the resource requirement of the AES algorithm.

Apart from the algorithmic optimizations like loop unrolling and splitting, the decision to choose appropriate resources of the target architecture also plays a significant role in the final performance of the implemented algorithm. For example, in this work, we choose Spartan-6 FPGA of Xilinx. This FPGA mainly uses Configurable Logic Blocks (CLBs) for the implementation of computing operations. The CLBs are generic computation blocks. Apart from CLBs, Spartan-6 FPGAs also have some dedicated blocks, which if chosen wisely can give significant advantages is performance and the overall resources requried for the implementation. Careful analysis of the different computation operation of AES algorithm indicates that operations like AddRoundkey, MixColumn, etc. can be implemented using CLBs only. However, the SubByte operation that involves S-box can either be implemented using CLBs or Block RAMs (BRAMs). It is clear from the discussion presented in previous section that the S-box values are predefined and they can be stored in BRAMs at the configuration time or they can also be stored in CLBs as CLBs can act both as computation blocks or the storage blocks. The usage of CLBs as storage blocks for S-box values can significantly shift the balance of AES algorithm implementation either in favor of performance or the resource requirements. In the following part of this section, we use different combinations of the algorithmic and architecture optimizations and explore their effect on the design of different implementation techniques.

Based on the discussion presented above, we have explored five different implementation techniques for AES algorithm. Due to the different algorithmic and architecture optimizations, these techniques give different resource and performance results. An overview of the implementation of these techniques is provided next.

***Technique 1:*** In the first technique, the S-Box for both cipher module and key expansion module is implemented in the BRAMs of target architecture. Moreover, the both the key expansion module and cipher module are executed in a serialized manner. In this manner, first the key is expanded and next the cipher module is executed. In terms of implementation, this is the simplest of the five techniques that we explore in this work. As the whole implementation is executed in a serialized manner, this technique gives us the best results in terms of resource requirement. However, the performance of this technique is quite low. This kind of technique is well suited for embedded devices with low resource availability and no performance constraints. But, it is not suitable for devices who are performance critical.

***Technique 2:*** Just like the first technique, in this technique, the S-box for both cipher module and key expansion module is implemented in BRAMs. However, contrary to first technique, here the two modules are executed in parallel. The parallel execution is achieved through loop unrolling in cipher module where N iterations of cipher module are unrolled and key generation through key expansion module is performed online through splitting. Because of the parallel execution, this technique greatly improves the critical path delay of the implementation. however, it may require significantly more resources as compared to the serialized implementation.

***Technique 3:*** In this technique, the S-box for cipher module is implemented in BRAMs whereas the entire key expansion module is implemented using CLBs. Moreover, the execution if the implementation is performed in a serialized way. That means, first the key expansion module is executed and they key is generated and next the cipher module is executed where generated keys are used for encryption/decryption. Compared to the first two techniques, this technique requires smaller number of BRAMs because of the key expansion module's S-box implementation in CLBs. This fact may also lead to better delay results as less number of BRAMs are involved in the critical path of the implementation.

***Technique 4:*** In this technique, the S-box for cipher module is implemented in BRAMs whereas the entire key expansion module is implemented using CLBs. Compared to technique 3, this technique is executed in a parallelized manner. The parallelism is achieved through loop unrolling and online key generation. Compared to technique 3, this techniqu gives better delay results but poor area results.

***Technique 5:*** In the last technique that we explore, both the cipher module and key expansion modules are implemented entirely using CLBs. Furthermore, both modules are executed in a serialized manner. Implementation of S-box in CLBs leads to very good delay results. However, this implementation results in very high number of CLBs that are required for the implementation.

In this section, we have given an overview of the different optimizations used for the exploration of different implementations. Further discussion on the results obtained for these implementation techniques is presented in Section IV.

## IV. Results and Analysis

### A. Experimental Setup

The five exploration techniques described in previous section are implemented on a Spartan-6 FPGA from Xilinx. For

TABLE I. EXPERIMENTAL RESULTS

| Technique | Number of Slice Register | Number of Slice LUTs | Frequency (MHz) | Throughput (Gb/S) | Efficiency |
|---|---|---|---|---|---|
| Technique 1 | 278 | 3315 | 137.29 | 17.57 | 4.85 |
| Technique 2 | 1547 | 3253 | 223.03 | 28.54 | 5.89 |
| Technique 3 | 280 | 4307 | 207.74 | 26.6 | 5.78 |
| Technique 4 | 1589 | 4530 | 214.96 | 27.51 | 4.51 |
| Technique 5 | 256 | 9375 | 886.64 | 113.49 | 11.78 |

this purpose, we have used xc6s1x150-3-fgg900 platform and the techniques are implemented using Xilinx's Vivado design suite. The VHDL core of each technique is synthesized, placed, and routed using this suite where explicit directives are used to ensure the implementation of S-box in either BRAMs or in CLBs. Moreover, parallel processes are used to ensure where parallel execution is needed. The synthesized implementation was used to measure a number of parameters pertaining to each implementation. These parameters include number of slice registers, slice LUTs (a term used alternatively for CLBs here), maximum frequency, and critical path delay etc. Moreover, theoretical throughput and efficiency of each implementation technique are also calculated using (2) and (3) respectively. Although the results presented in this work are based on a Xilinx FPGA, the optimization techniques are generic in nature and they are applicable to any underlying hardware. A thorough discussion on the results of each technique is provided next.

$$T_{\text{put}} = \frac{Processed\ bits}{Delay} \tag{2}$$

$$Efficiency = \frac{T_{\text{put}}}{Resources} \tag{3}$$

### B. Experimental Results

Experimental results obtained after the implementation of five exploration techniques are given in Table I. In this table, the first column corresponds to the technique while next five columns indicate the number of slice registers, slice LUTs, frequency, throughput, and efficiency values obtained for each technique.

It can be seen from second column of Table I that the techniques using parallelism (i.e., Technique 2 and 4) require significantly more slice registers as compared to the techniques that are implemented in a serialized manner. This is because of the fact that while parallelizing the implementation, significantly more registers are required for each stage of cipher module and key expansion module. These registers are used to keep the two modules in complete synchronization and without them it will not possible to parallelize the implementation. The next column gives a comparison of slice LUTs for different exploration techniques. It is clear from the results presented in column three that the techniques using BRAMs for their S-box implementation require less number of slice LUTs as compared to the techniques using CLB (or LUTs) for the implementation of S-box. Furthermore, it can also be observed from third column that technique 5 requires significantly more slice LUTs than any other technique. This is because of the reason that this technique is implementing the S-box of both cipher module and key expansion module in LUTs.

The routing of each exploration technique also gives its corresponding operating frequency. The frequency numbers of each technique are given in the fourth column of Table I. It can be seen from this table that in general the techniques employing parallelism have higher frequency and the techniques implemented in a serialized manner have a lower operating frequency. There is one exception however. Technique 5 is implemented entirely in a serialized manner, but still it reports high frequency results. This is because of the fact that complete absence of BRAMs reduces the internal delay of BRAMs and also eliminates the communication delay between CLBs and BRAMS eventually resulting in better frequency results as compared to all other techniques. We further consolidate the frequency and critical path delay in the form of Figure 2. In this figure, solid and dashed lines indicate the frequency and critical path delay results respectively. It is clear from this figure that the techniques with high frequency results hav lower critical path delay and vice versa.

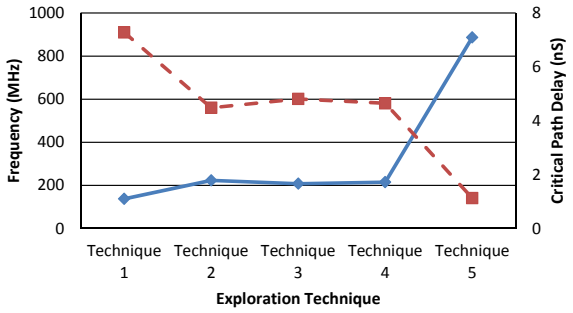


Figure 2. Frequency-Delay Comparison

We have also computed the theoretical throughput results using (2) and these results are depicted in the column 5 of Table I. It can be seen from these numbers that in general the techniques with higher frequency have better throughput as compared to the techniques with lower frequency. The efficiency results of each technique are also computed using (3) and they are depicted in column 6 of Table I. It can be seen from this table that technique 5 gives the best efficiency results. This is mainly because of the reason that this technique uses less number of registers and almost no BRAMs. Moreover, this technique gives significantly better frequency results as compared to all other techniques which eventually leads to the best efficiency results for this technique.

Finally, to have a complete overview of the quality of an implementation, we perform a comparison between the total resource requirement and the critical path delay numbers of all the techniques under consideration. In this comparison, the resource requirement gives an overview of the area and critical path delay number gives an overview of the performance of

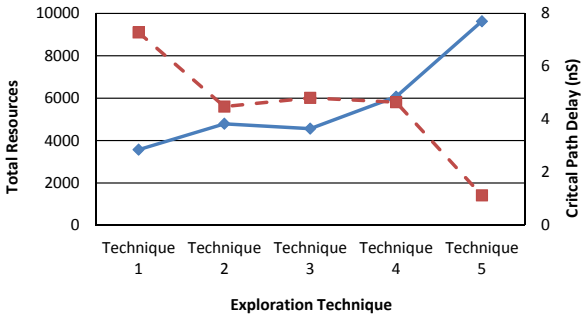


Figure 3. Area-Delay Comparison

the technique. These results are shown in Figure 3. In this figure, the resources are shown as solid line while critical path delay values are shown as dashed line. It can be seen from this figure that technique 1 requires smallest number of resources; hence it is suitable for area constrained IoT devices. However, it should be noted that this technique also has the poorest critical path delay value. So, this kind of technique is suitable to secure edge side nodes. On the other hand, technique 5 requires highest number of total resources but at the same time it gives the best delay results as well. From these numbers, it can be concluded that this kind of technique is well suited for devices that are performance constrained and where number of resources is not an issue for them.

## V. Conclusion

IoT devices have gained tremendous popularity over the past few years and they are now the driving force of a multi-billion dollar industry. Modern IoT devices are quite heterogeneous in nature and they are subject to all sorts of security threats. In this work, based on various algorithmic and architecture optimizations, we explore five different implementations of AES cryptographic scheme. We consider AES as it is quite robust and its five different implementations are well suited for the varying requirements of heterogeneous IoT devices. Experimental results of these implementations reveal that the serialized implementation of cipher and key expansion modules of AES algorithms leads to the best area results; hence making the serialized implementation suitable for resource constrained edge IoT devices. However, this kind of implementation is not suitable for high performance IoT devices. For such devices, the experimental results reveal that the implementations using parallelism are more suitable. Although such implementations are quite resource hungry, they give very good performance results.

## References

[1] D. Singh, G. Tripathi, and A. J. Jara, "A survey of internet-of-things: Future vision, architecture, challenges and services," in 2014 IEEE World Forum on Internet of Things (WF-IoT), March 2014, pp. 287–292.

[2] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," IEEE Transactions on Emerging Topics in Computing, vol. 5, no. 4, Oct 2017, pp. 586–602.

[3] M. Zhang, A. Raghunathan, and N. K. Jha, "Trustworthiness of medical devices and body area networks," Proceedings of the IEEE, vol. 102, no. 8, Aug 2014, pp. 1174–1188.

[4] K. E. Psannis, C. Stergiou, and B. B. Gupta, "Advanced media-based smart big data on intelligent cloud systems," IEEE Transactions on Sustainable Computing, vol. 4, no. 1, 2018, pp. 77–87.

[5] Y. Cherdantseva and J. Hilton, "A reference model of information assurance amp; security," in 2013 International Conference on Availability, Reliability and Security, Sept 2013, pp. 546–555.

[6] S. Vashi, J. Ram, J. Modi, S. Verma, and C. Prakash, "Internet of things (iot): A vision, architectural elements, and security issues," in 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Feb 2017, pp. 492–496.

[7] M. M. Kermani, M. Zhang, A. Raghunathan, and N. K. Jha, "Emerging frontiers in embedded security," in 2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems, Jan 2013, pp. 203–208.

[8] H. Salmani and M. M. Tehranipoor, "Vulnerability analysis of a circuit layout to hardware trojan insertion," IEEE Transactions on Information Forensics and Security, vol. 11, no. 6, June 2016, pp. 1214–1225.

[9] H. Suo, J. Wan, C. Zou, and J. Liu, "Security in the internet of things: A review," in 2012 International Conference on Computer Science and Electronics Engineering, vol. 3, March 2012, pp. 648–651.

[10] K. Hu, A. N. Nowroz, S. Reda, and F. Koushanfar, "High-sensitivity hardware trojan detection using multimodal characterization," in 2013 Design, Automation Test in Europe Conference Exhibition (DATE), March 2013, pp. 1271–1276.

[11] G. Rouvroy, F. X. Standaert, J.-J. Quisquater, and J. Legat, "Compact and efficient encryption/decryption module for fpga implementation of the aes rijndael very well suited for small embedded applications," in Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on, vol. 2, April 2004, pp. 583–587 Vol.2.

[12] J. Van Dyken and J. G. Delgado-Frias, "Fpga schemes for minimizing the power-throughput trade-off in executing the advanced encryption standard algorithm," J. Syst. Archit., vol. 56, no. 2-3, Feb. 2010, pp. 116–123. [Online]. Available: http://dx.doi.org/10.1016/j.sysarc.2009.12.001

[13] T. Hoang and V. L. Nguyen, "An efficient fpga implementation of the advanced encryption standard algorithm," in Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on, Feb 2012, pp. 1–4.

[14] M. I. Soliman and G. Y. Abozaid, "{FPGA} implementation and performance evaluation of a high throughput crypto coprocessor," Journal of Parallel and Distributed Computing, vol. 71, no. 8, 2011, pp. 1075 – 1084.

[15] A. Gielata, P. Russek, and K. Wiatr, "Aes hardware implementation in fpga for algorithm acceleration purpose," in Signals and Electronic Systems, 2008. ICSES '08. International Conference on, Sept 2008, pp. 137–140.

[16] S. Qu, G. Shou, Y. Hu, Z. Guo, and Z. Qian, "High throughput, pipelined implementation of aes on fpga," in Information Engineering and Electronic Commerce, 2009. IEEC '09. International Symposium on, May 2009, pp. 542–545.

[17] S. Kulkarni, S. Durg, and N. Iyer, "Internet of things (iot) security," in Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on. IEEE, 2016, pp. 821–824.

[18] J. Daemen and V. Rijmen, The Block Cipher Rijndael, ser. Lecture Notes in Computer Science, J.-J. Quisquater and B. Schneier, Eds. Springer Berlin Heidelberg, 2000, vol. 1820.

 ISBN: 978-1-61208-818-1 